Geant4 11.1-p02

Physics Overview

10th International Geant4 Tutorial in Korea

Dennis Wright

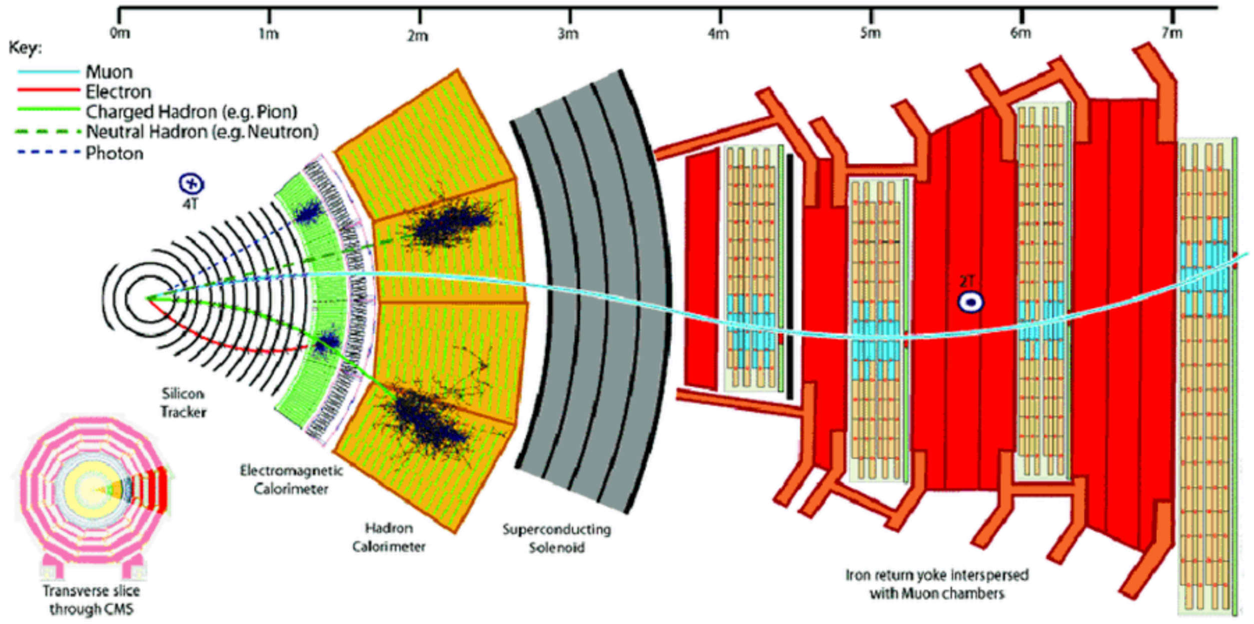
6-10 November 2023

Outline

- Introduction - Geant4 physics
- Physics lists
  - What is a physics list? Why do we need it?
  - G4VUserPhysicsList
  - Modular physics lists - a more convenient way to go
  - Pre-packaged physics lists - provided by the toolkit
  - Examples
- Geant4 physics overview
  - EM, hadronic, decay, transportation
- Processes
  - What is a process and what does it do?

Introduction

- Geant4 provides a variety of physics simulation components over a wide energy range and a wide range of particles
  - Primary generation → event → particles
  - Detector description → geometry and material → target nuclei
  - Physics list interaction → (particle, target energy) → final state


0m
1m
2m
3m
4m
5m
6m
7m
Key:
Muon
Electron
Charged Hadron (e.g. Pion)
Neutral Hadron (e.g. Neutron)
Photon
4T
2T
Silicon Tracker
Electromagnetic Calorimeter
Hadron Calorimeter
Superconducting Solenoid
Iron return yoke interspersed with Muon chambers
Transverse slice through CMS

What is a Physics List?

- An object responsible for:
  - specifying all particles to be used in a simulation application
  - specifying physics processes and assigning them to each particle type
- One of three mandatory objects that the user must provide to the G4RunManager in any application
  - tells run manager what physics needs to be invoked and when
- Provides a very flexible way to set up the physics environment
  - user can choose and specify particles he wants
  - user can choose the physics (processes) to assign to each particle
- BUT, user must have a good understanding of the physics required to describe the problem
  - omission of relevant particles and/or physics interactions could lead to poor modeling results

Why Do We Need a Physics List?

- Physics is physics – shouldn’t Geant4 provide, as a default, a complete set of physics that everyone can use?

- NO:
  - there are many different approximations and models to describe the same interaction
    - very much the case for hadronic but also for electromagnetic physics
  - computation time is an issue:
    - some users may want a less accurate but significantly faster model for a given interaction while others need the most accurate description regardless of CPU time
  - there is no simulation application that would require all the particles and all the possible interactions that Geant4 can provide
    - e.g. most medical applications are not interested in multi-GeV physics

- For this reason Geant4 takes an atomistic, rather than an integral approach to physics
  - provides many independent (for the most part) physics components (i.e. physics processes)
  - users select these components in their custom-designed physics lists
  - exceptions: a few electromagnetic processes must be used together

Physics Processes Provided by Geant4

- Electromagnetic physics
  - “standard”: the default processes valid between ~keV and PeV
  - ”low energy”: processes available for ~100 eV to 1 PeV
  - Geant4 DNA: valid down to ~eV (but mostly just for liquid water)
  - optical photons

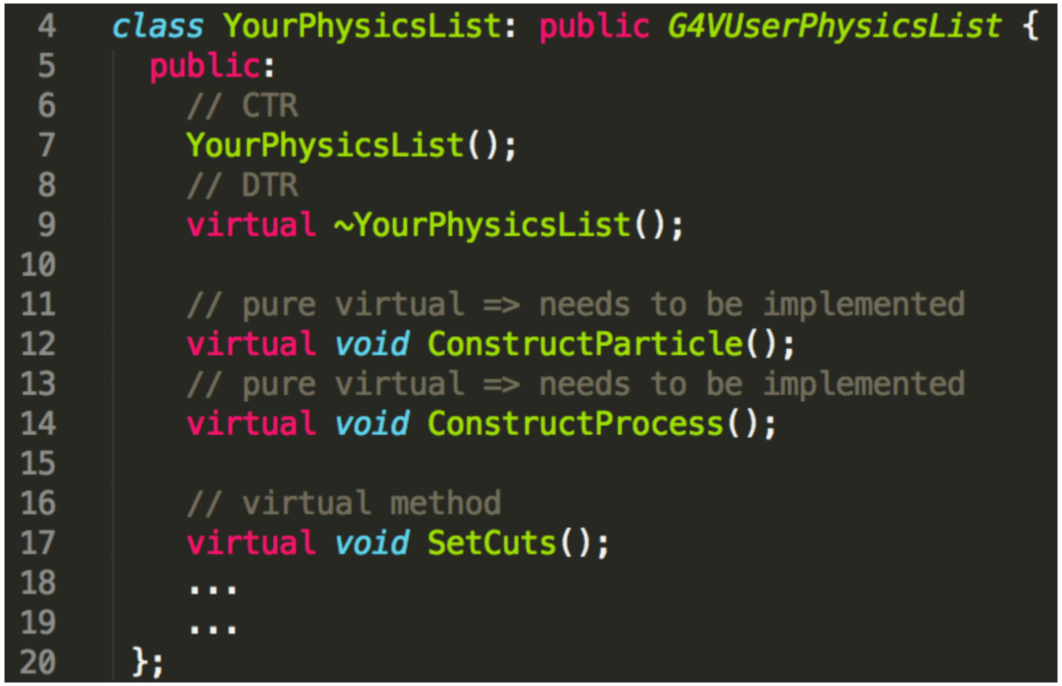
- Weak interactions
  - decay of subatomic particles
  - radioactive decay of nuclei

- Hadronic physics
  - pure strong interaction physics valid from 0 to ~1 TeV
  - electro- and gamma-nuclear interactions valid from 10 MeV to ~TeV
  - high precision neutron (and other particles) package valid from thermal energies to ~20 MeV

- Parameterized or “fast simulation” physics

Physics List Interface

- G4VUserPhysicsList is the Geant4 physics list interface
- All physics lists must derive from this base class



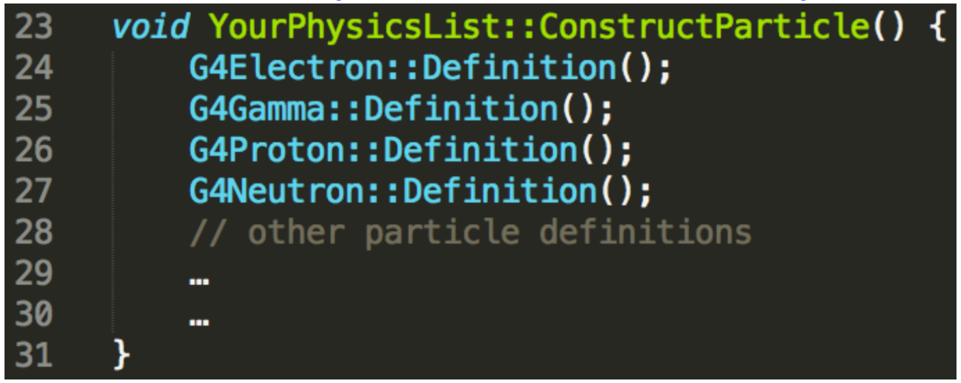
```
class YourPhysicsList: public G4VUserPhysicsList {
  public:
    // CTR
    YourPhysicsList();
    // DTR
    virtual ~YourPhysicsList();

    // pure virtual => needs to be implemented
    virtual void ConstructParticle();
    // pure virtual => needs to be implemented
    virtual void ConstructProcess();

    // virtual method
    virtual void SetCuts();
    ...
    ...
};
```

- User must implement the two pure virtual methods ConstructParticle() and ConstructProcess()
- User can implement the SetCuts() method (optional)

Physics List Interface: ConstructParticle()

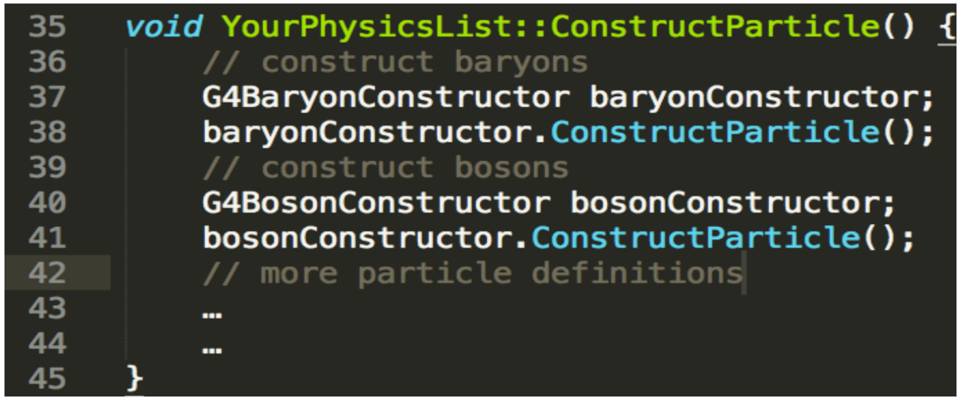
- Interface method defines list of particles to be used in the application

- Can construct particles individually



```
void YourPhysicsList::ConstructParticle() {
    G4Electron::Definition();
    G4Gamma::Definition();
    G4Proton::Definition();
    G4Neutron::Definition();
    // other particle definitions
    …
    …
}
```

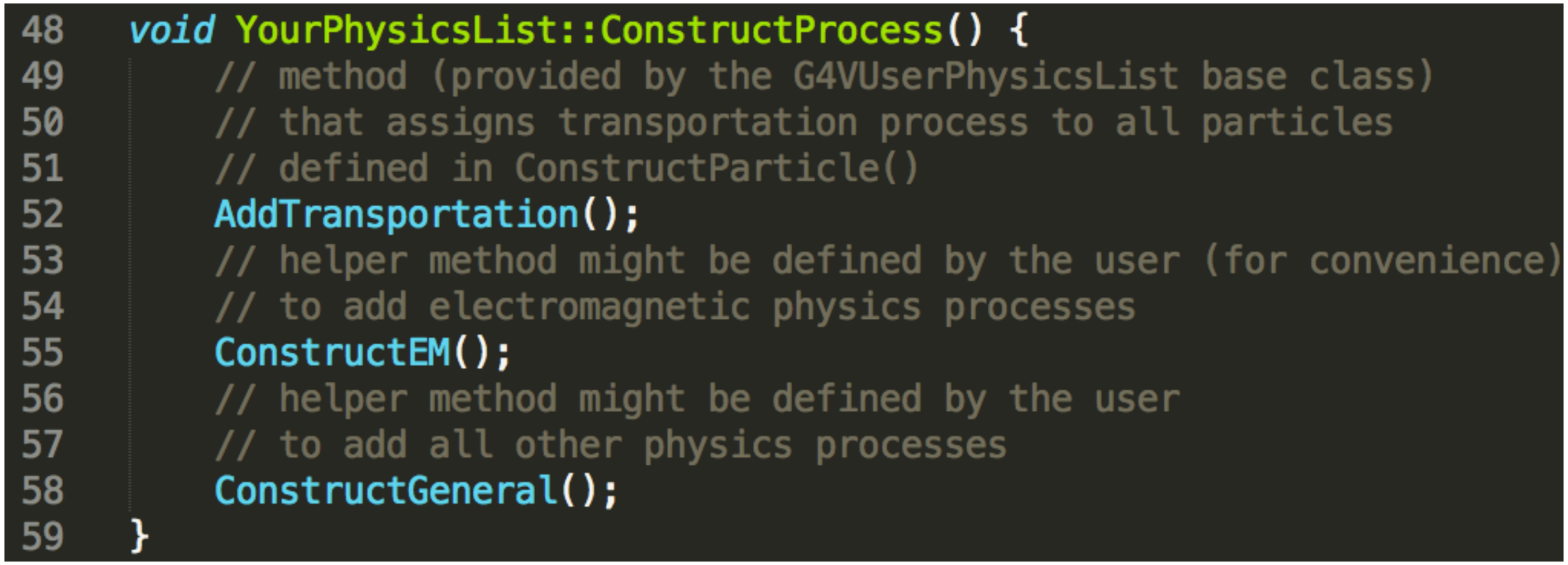
- Or using toolkit-provided helper classes



```
void YourPhysicsList::ConstructParticle() {
    // construct baryons
    G4BaryonConstructor baryonConstructor;
    baryonConstructor.ConstructParticle();
    // construct bosons
    G4BosonConstructor bosonConstructor;
    bosonConstructor.ConstructParticle();
    // more particle definitions
    …
    …
}
```

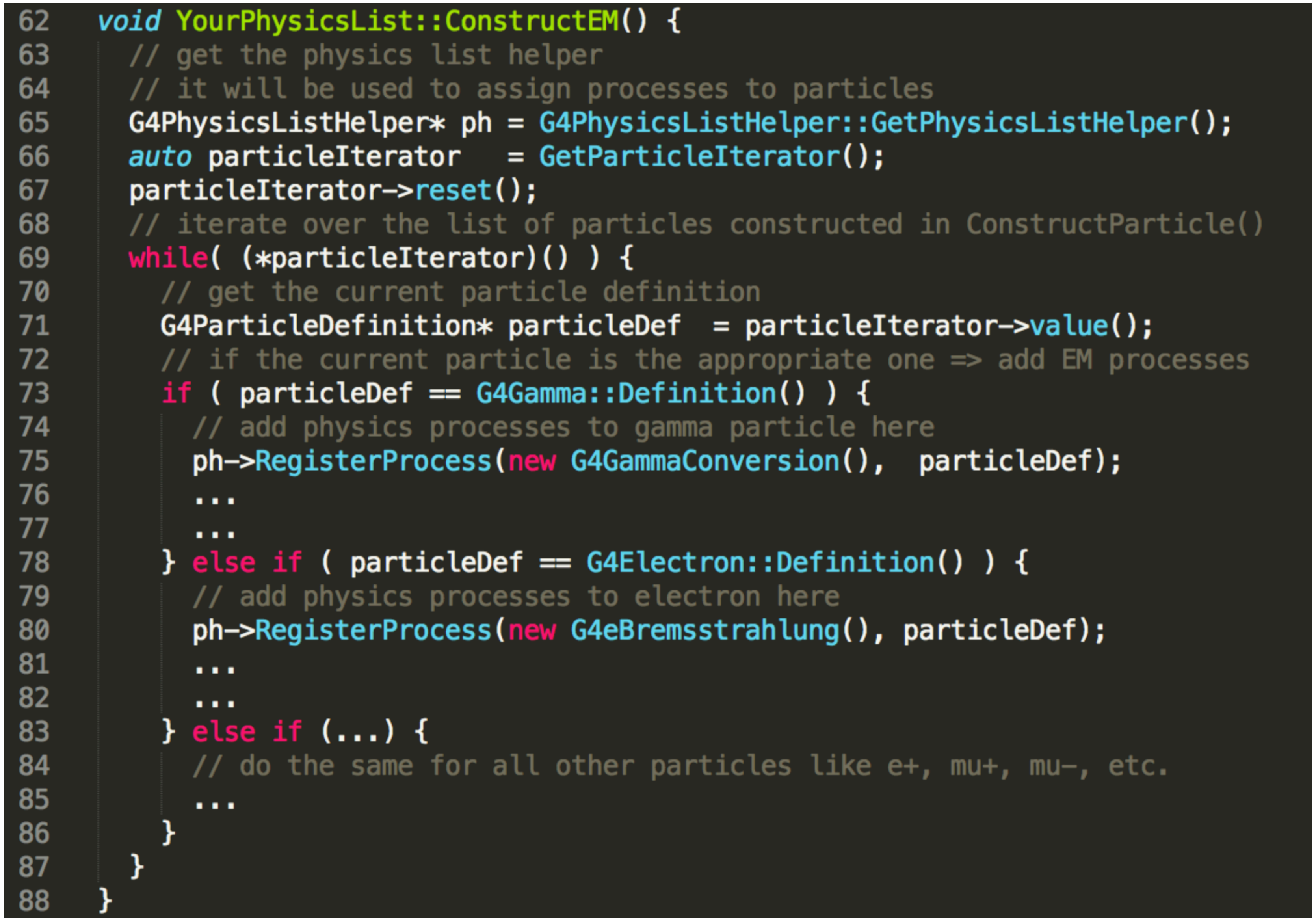
Physics List Interface: ConstructProcess()

- What is a process?
  - an object that defines the way in which a specific particle interacts with matter through a given type of interaction (e.g. electron ionization)
- Interface method: defines the list of physics processes to be used in the simulation for a given particle type



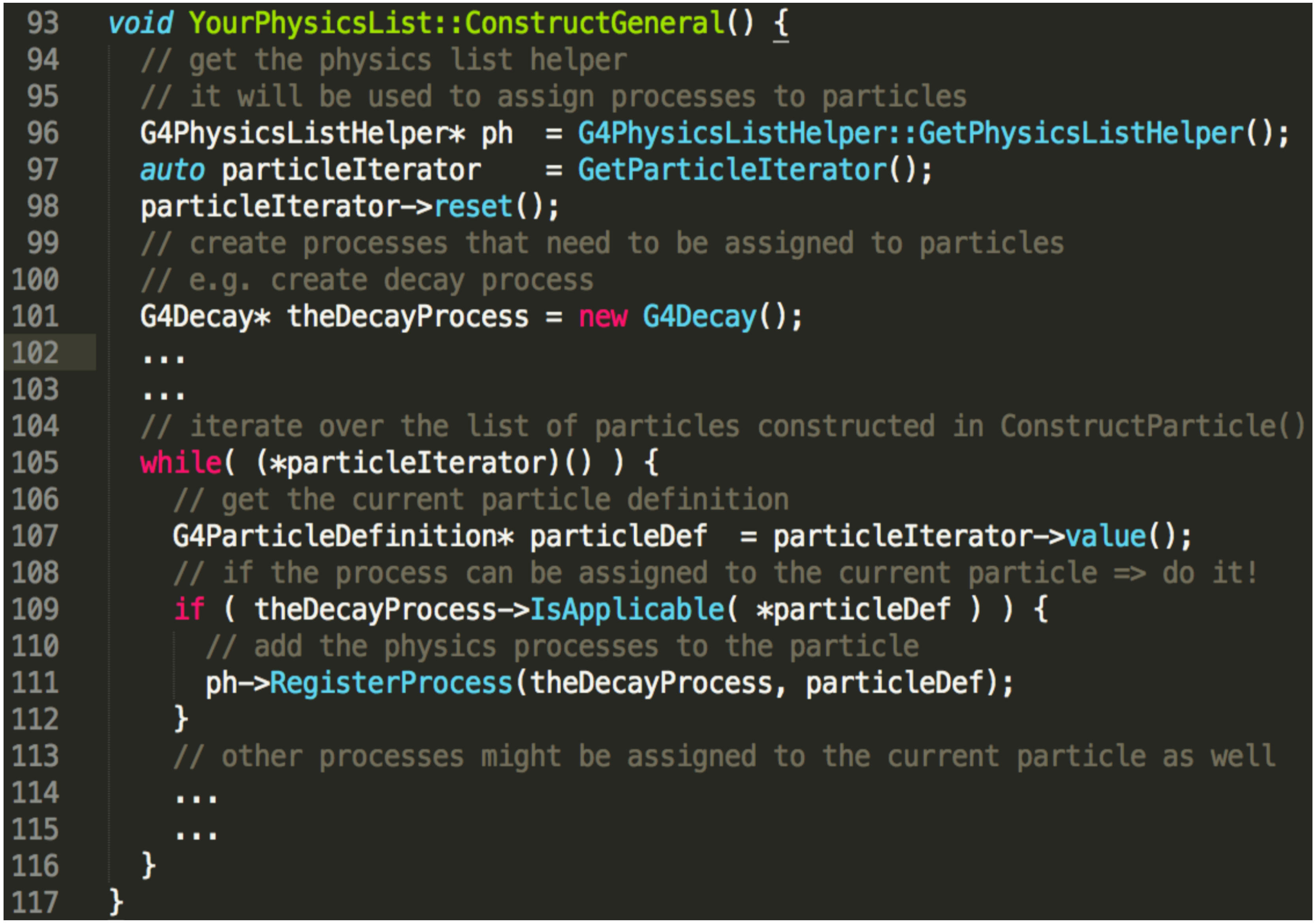
```
void YourPhysicsList::ConstructProcess() {
    // method (provided by the G4VUserPhysicsList base class)
    // that assigns transportation process to all particles
    // defined in ConstructParticle()
    AddTransportation();
    // helper method might be defined by the user (for convenience)
    // to add electromagnetic physics processes
    ConstructEM();
    // helper method might be defined by the user
    // to add all other physics processes
    ConstructGeneral();
}
```

Physics List Interface: ConstructProcess()



```
void YourPhysicsList::ConstructEM() {
  // get the physics list helper
  // it will be used to assign processes to particles
  G4PhysicsListHelper* ph = G4PhysicsListHelper::GetPhysicsListHelper();
  auto particleIterator   = GetParticleIterator();
  particleIterator->reset();
  // iterate over the list of particles constructed in ConstructParticle()
  while( (*particleIterator)() ) {
    // get the current particle definition
    G4ParticleDefinition* particleDef  = particleIterator->value();
    // if the current particle is the appropriate one => add EM processes
    if ( particleDef == G4Gamma::Definition() ) {
      // add physics processes to gamma particle here
      ph->RegisterProcess(new G4GammaConversion(),  particleDef);
      ...
      ...
    } else if ( particleDef == G4Electron::Definition() ) {
      // add physics processes to electron here
      ph->RegisterProcess(new G4eBremsstrahlung(), particleDef);
      ...
      ...
    } else if (...) {
      // do the same for all other particles like e+, mu+, mu-, etc.
      ...
    }
  }
}
```

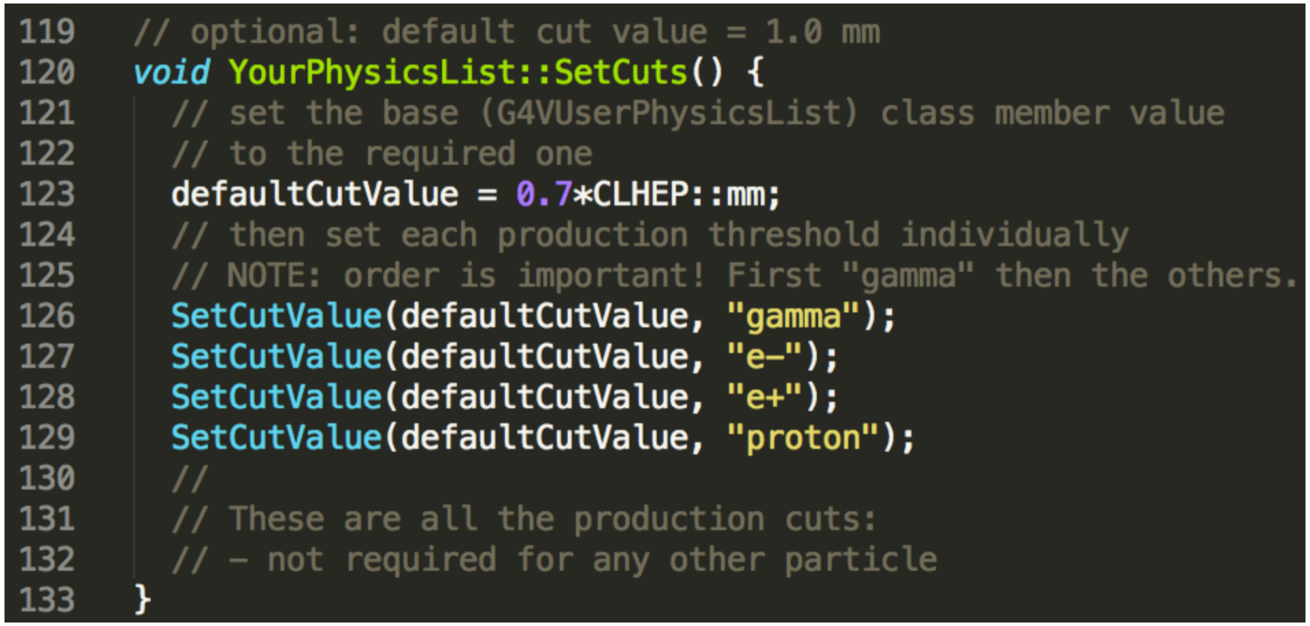
Physics List Interface: ConstructProcess()



```cpp
void YourPhysicsList::ConstructGeneral() {
  // get the physics list helper
  // it will be used to assign processes to particles
  G4PhysicsListHelper* ph  = G4PhysicsListHelper::GetPhysicsListHelper();
  auto particleIterator    = GetParticleIterator();
  particleIterator->reset();
  // create processes that need to be assigned to particles
  // e.g. create decay process
  G4Decay* theDecayProcess = new G4Decay();
  ...
  ...
  // iterate over the list of particles constructed in ConstructParticle()
  while( (*particleIterator)() ) {
    // get the current particle definition
    G4ParticleDefinition* particleDef  = particleIterator->value();
    // if the process can be assigned to the current particle => do it!
    if ( theDecayProcess->IsApplicable( *particleDef ) ) {
      // add the physics processes to the particle
      ph->RegisterProcess(theDecayProcess, particleDef);
    }
    // other processes might be assigned to the current particle as well
    ...
    ...
  }
}
```

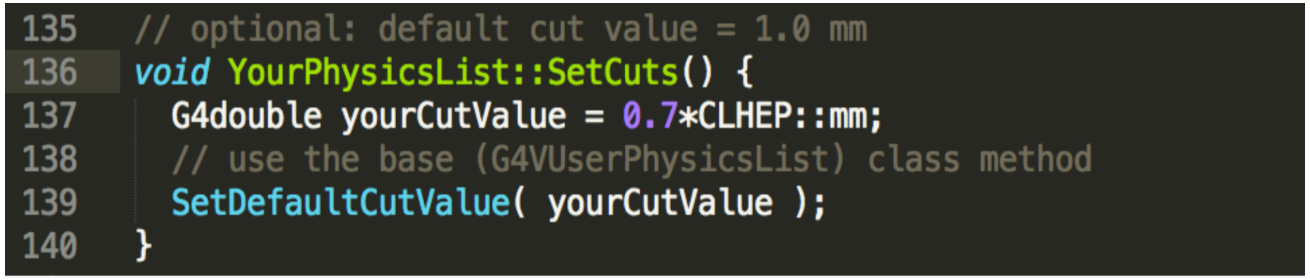
Physics List Interface: SetCuts()

- Interface method (optional):



```
// optional: default cut value = 1.0 mm
void YourPhysicsList::SetCuts() {
  // set the base (G4VUserPhysicsList) class member value
  // to the required one
  defaultCutValue = 0.7*CLHEP::mm;
  // then set each production threshold individually
  // NOTE: order is important! First "gamma" then the others.
  SetCutValue(defaultCutValue, "gamma");
  SetCutValue(defaultCutValue, "e-");
  SetCutValue(defaultCutValue, "e+");
  SetCutValue(defaultCutValue, "proton");
  //
  // These are all the production cuts:
  // - not required for any other particle
}
```

- Or a simpler (and equivalent) way:



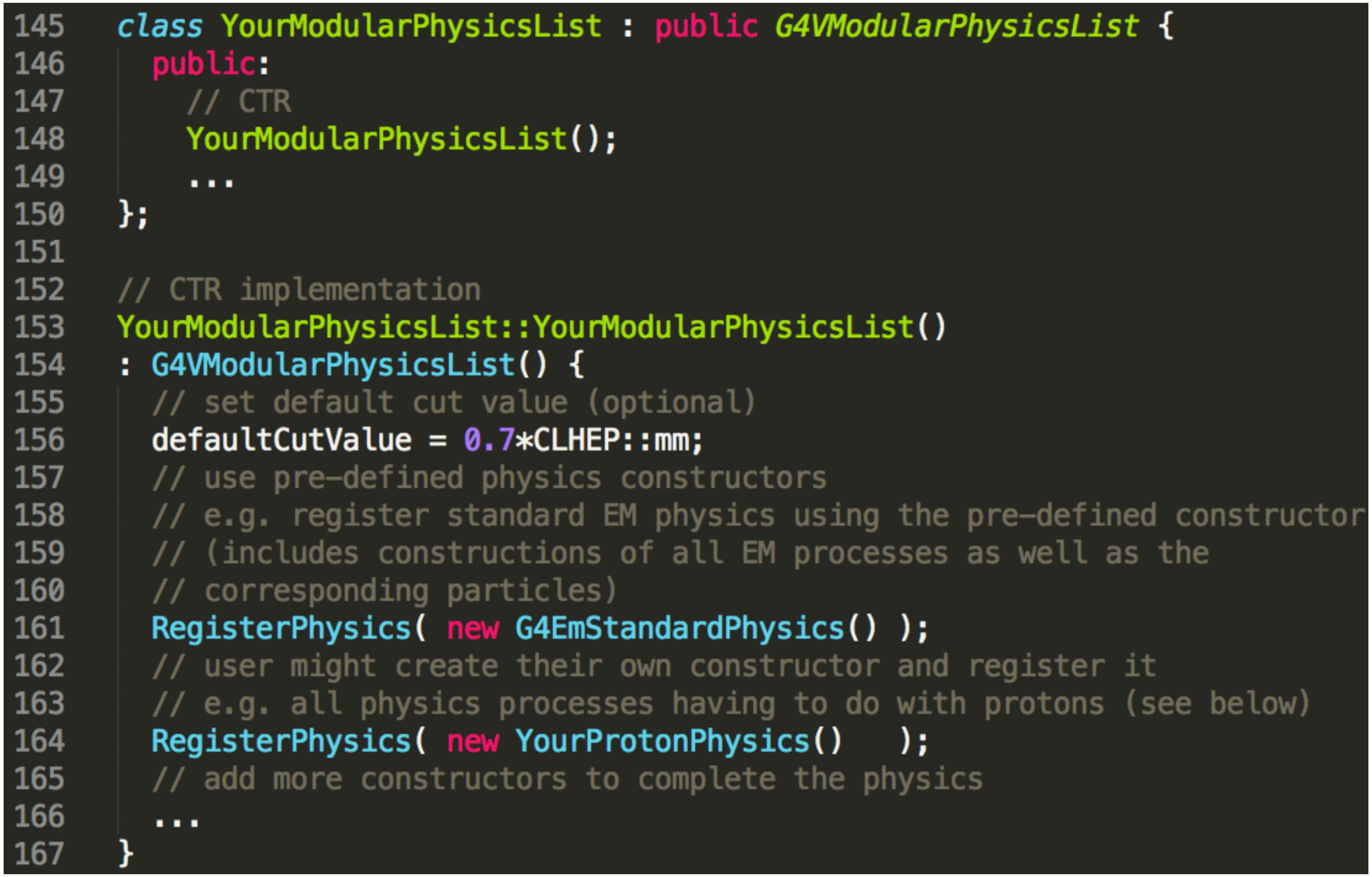
```
// optional: default cut value = 1.0 mm
void YourPhysicsList::SetCuts() {
  G4double yourCutValue = 0.7*CLHEP::mm;
  // use the base (G4VUserPhysicsList) class method
  SetDefaultCutValue( yourCutValue );
}
```

Modular Physics List

- Why use this?
  - previous physics list example was very simple and incomplete
  - realistic physics lists will have many more particles and processes
  - such a list can be quite long, complicated and hard to maintain

- Modular physics list provides a solution:
  - interface is defined in G4VModularPhysicsList
  - this interface is derived from the G4VUserPhysicsList base class (as YourPhysicsList in the previous example)
  - the transportation process is automatically added to all constructed particles
  - allows the use of "physics modules"
  - a given physics module handles a well-defined category of physics e.g. EM physics, hadronic physics, decay, etc.

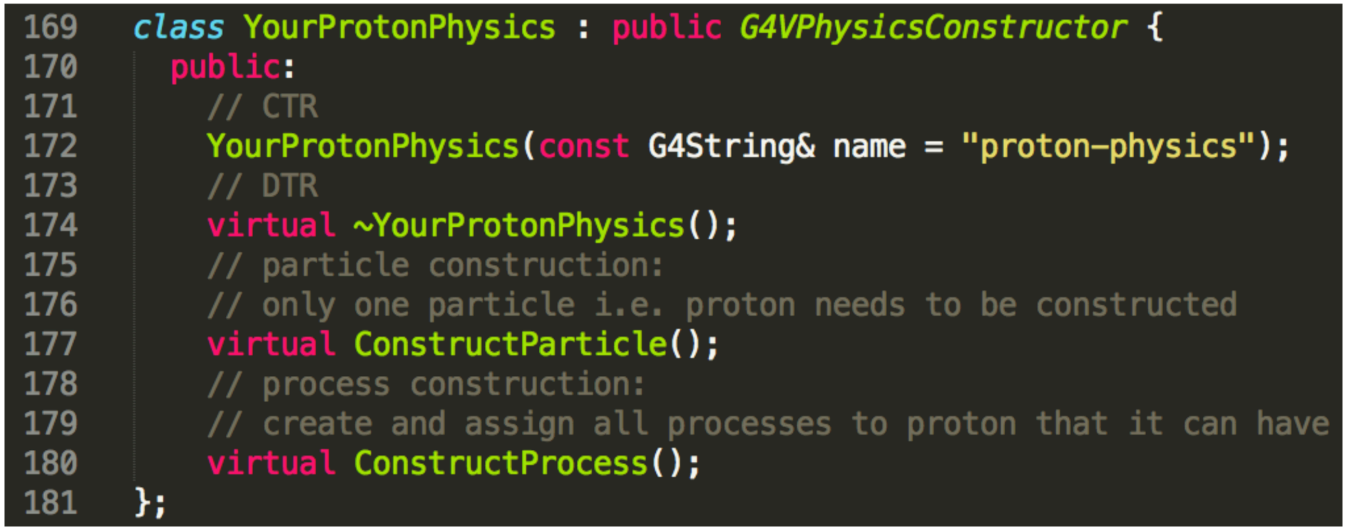
Modular Physics List



```cpp
class YourModularPhysicsList : public G4VModularPhysicsList {
  public:
    // CTR
    YourModularPhysicsList();
    ...
};

// CTR implementation
YourModularPhysicsList::YourModularPhysicsList()
: G4VModularPhysicsList() {
  // set default cut value (optional)
  defaultCutValue = 0.7*CLHEP::mm;
  // use pre-defined physics constructors
  // e.g. register standard EM physics using the pre-defined constructor
  // (includes constructions of all EM processes as well as the
  // corresponding particles)
  RegisterPhysics( new G4EmStandardPhysics() );
  // user might create their own constructor and register it
  // e.g. all physics processes having to do with protons (see below)
  RegisterPhysics( new YourProtonPhysics()   );
  // add more constructors to complete the physics
  ...
}
```

Modular Physics List: Physics Constructors

- Physics constructor
  - allows particles and their associated processes to be grouped together according to a physics domain
  - implements the G4VPhysicsConstructor
  - can be viewed as a subset of a complete physics list
  - user may create his own (e.g. YourProtonPhysics) or use pre-defined physics constructors (G4EmStandardPhysics, G4DecayPhysics, …)



```
class YourProtonPhysics : public G4VPhysicsConstructor {
  public:
    // CTR
    YourProtonPhysics(const G4String& name = "proton-physics");
    // DTR
    virtual ~YourProtonPhysics();
    // particle construction:
    // only one particle i.e. proton needs to be constructed
    virtual ConstructParticle();
    // process construction:
    // create and assign all processes to proton that it can have
    virtual ConstructProcess();
};
```

Packaged Physics Lists

- Our examples dealt mainly with EM physics
- A realistic physics list is found in basic example B3
  - modular physics list including standard EM physics and decay physics built with physics constructors
  - good starting point to construct your own physics list
  - add other physics to suit your needs
- For both hadronic and electromagnetic physics, processes may be implemented by “models”
  - user may choose from several “models”
  - choosing the most appropriate model for a given application requires significant experience
- Pre-packaged physics lists
  - provided by toolkit and developed for a few reference cases
  - ready-to-use, developed by experts in certain application areas
  - each pre-packaged list contains different combinations of EM and hadronic physics
  - list of these found in toolkit at geant4/source/physics_lists/lists/include

Packaged Physics Lists

- Caveats:
    - these lists are provided as a best guess of the physics needed in some given use cases
    - user is responsible for validating the particular physics list for a given application and adding or removing physics if necessary
    - intended as starting points or templates

- Production physics lists
    - used by a large user groups such as ATLAS and CMS
    - well-maintained and tested
    - very stable: fewer changes, less frequent updates

- Pre-packaged physics lists
    - provided by toolkit and developed for a few reference cases
    - ready-to-use, developed by experts in certain application areas
    - extensively validated by developers and the user communities
    - FTFP_BERT, QGSP_BERT, QGSP_FTFP_BERT_EMV, FTFP_BERT_HP, …

Packaged Physics Lists: Naming Convention

- Hadronic options
  - QGS – quark gluon string model (> ~15 GeV)
  - FTF – FRITIOF QCD string model (> ~ 5 GeV)
  - BERT – Bertini cascade (< ~12 GeV)
  - BIC – Binary interaction cascade (< ~ 10 GeV)
  - P – G4Precompound deexcitation model
  - HP – high precision neutron, proton, d, t, ^{3}He, alpha interaction model (< 20 MeV)

- Electromagnetic options
  - no suffix – standard EM physics (the default G4EmStandardPhysics constructor)
  - EMV – G4EmStandardPhysics_option1 (HEP, fast but less precise)
  - EMY – G4EmStandardPhysics_option3 (tuned for medical, space applications)
  - EMZ – G4EmStandardPhysics_option4 (most precise EM physics, slower)

- Name decoding: string_cascade_neutron_EM
- Complete list of pre-packaged physics lists with detailed descriptions in "Guide for Physics Lists" : geant4-userdoc.web.cern.ch/geant4-userdoc/UsersGuides/PhysicsListGuide/html/index.html

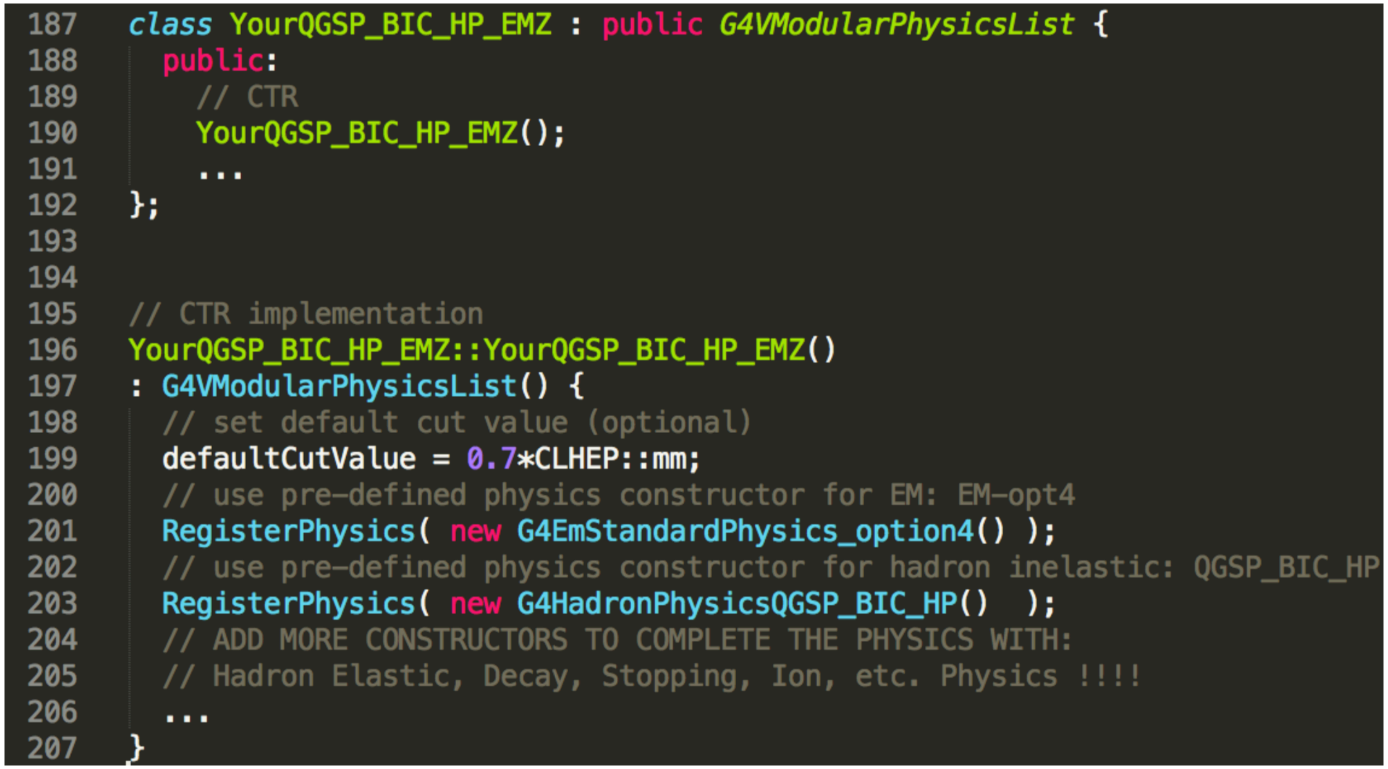
Packaged Physics Lists: Naming Convention Examples

- FTFP_BERT
  - includes standard EM physics models
  - FTF – FRITIOF string model (> ~4 GeV) +
  - P – G4Precompound deexcitation model
  - Bertini cascade (< ~12 GeV)

- QGSP_BIC_HP
  - QGS – quark gluon string model (> 12 GeV)
  - FTF – FRITIOF string model (9.5 - 25 GeV)
  - P – G4Precompound deexcitation model
  - BIC – Binary interaction cascade (200 MeV – 9.9 GeV)
  - HP – high precision neutron, proton, d, t, ^{3}He, alpha interaction model (< 20 MeV)

Example Using Physics Constructors

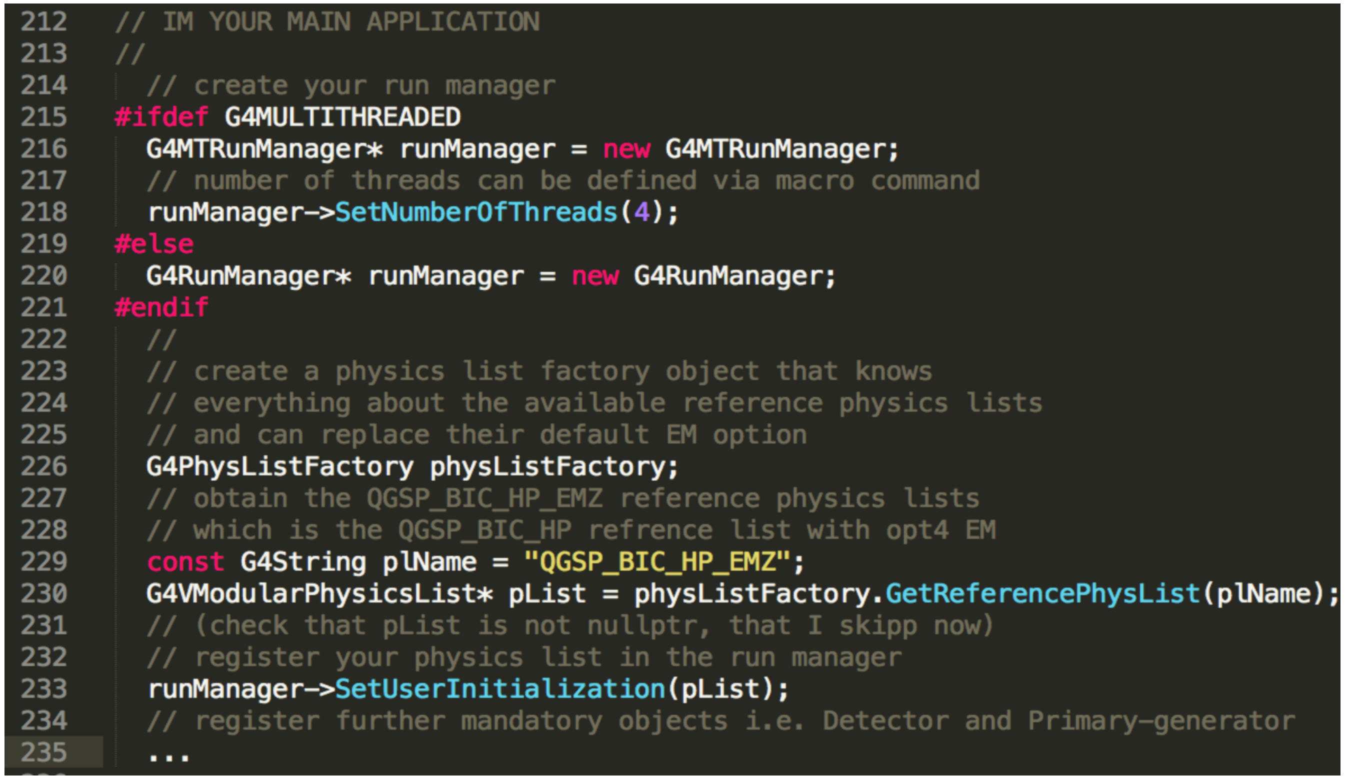
- QGSP_BIC_HP_EMZ
  - not currently a packaged list – we're going to to build it here
  - using constructors G4HadronPhysicsQGSP_BIC_HP and G4EmStandard_option4 (EMZ)



```
class YourQGSP_BIC_HP_EMZ : public G4VModularPhysicsList {
  public:
    // CTR
    YourQGSP_BIC_HP_EMZ();
    ...
};


// CTR implementation
YourQGSP_BIC_HP_EMZ::YourQGSP_BIC_HP_EMZ()
: G4VModularPhysicsList() {
  // set default cut value (optional)
  defaultCutValue = 0.7*CLHEP::mm;
  // use pre-defined physics constructor for EM: EM-opt4
  RegisterPhysics( new G4EmStandardPhysics_option4() );
  // use pre-defined physics constructor for hadron inelastic: QGSP_BIC_HP
  RegisterPhysics( new G4HadronPhysicsQGSP_BIC_HP()  );
  // ADD MORE CONSTRUCTORS TO COMPLETE THE PHYSICS WITH:
  // Hadron Elastic, Decay, Stopping, Ion, etc. Physics !!!!
  ...
}
```

Example Using Reference Physics Lists

- QGSP_BIC_HP_EMZ
  - the QGSP_BIC_HP reference physics list includes all the above physics constructors (but with standard EM physics)
  - G4PhysicsListFactory knows about all the available reference lists and makes possible the replacement of one EM option with another



```
// IM YOUR MAIN APPLICATION
//
  // create your run manager
#ifdef G4MULTITHREADED
  G4MTRunManager* runManager = new G4MTRunManager;
  // number of threads can be defined via macro command
  runManager->SetNumberOfThreads(4);
#else
  G4RunManager* runManager = new G4RunManager;
#endif
  //
  // create a physics list factory object that knows
  // everything about the available reference physics lists
  // and can replace their default EM option
  G4PhysListFactory physListFactory;
  // obtain the QGSP_BIC_HP_EMZ reference physics lists
  // which is the QGSP_BIC_HP refrence list with opt4 EM
  const G4String plName = "QGSP_BIC_HP_EMZ";
  G4VModularPhysicsList* pList = physListFactory.GetReferencePhysList(plName);
  // (check that pList is not nullptr, that I skipp now)
  // register your physics list in the run manager
  runManager->SetUserInitialization(pList);
  // register further mandatory objects i.e. Detector and Primary-generator
  ...
```

Geant4 Physics

- Physics components are coded as processes
  - a process is a well-defined interaction of a particle with matter
  - determines when the interaction happens and what the result is
  - Geant4 provides a large number of these
  - users may write their own, but they must be derived from a Geant4 process
  - all processes are developed by implementing the interface G4VProcess

- Processes are classified as
  - electromagnetic, hadronic, decay, parameterized or transportation

Geant4 Physics: Electromagnetic

- Standard – complete set of processes covering charged particles and gammas
    - energy range 1 keV to ~PeV

- Low energy – specialized routines for e^- , γ, charged hadrons
    - more atomic shell structure details
    - some processes valid down to 250 eV or below
    - others not valid above a few GeV

- Optical photons – only for long wavelength photons (x-rays, UV, visible)
    - processes for reflection/refraction, absorption, wavelength shifting, Rayleigh scattering
    - users select these components in their physics lists

Geant4 Physics: Hadronic

- Pure hadronic (0 to ~TeV)
  - elastic
  - inelastic
  - capture
  - fission

- Radioactive decay
  - at rest and in-flight

- Photo-nuclear (~10 MeV - ~TeV)
  - gamma-induced nuclear reactions

- Lepto-nuclear (~10 MeV - ~TeV)
  - e^+, e^- induced nuclear reactions
  - muon-induced nuclear reactions

Physics Processes:
Decay, Parameterized and Transportation

- Decay processes include
  - weak decay (leptonic, semi-leptonic decays, radioactive decay of nuclei)
  - electromagnetic decay (π^0 , Σ^0 , etc.)
  - strong decays not included here (they are part of hadronic models)

- Parameterized process
  - electromagnetic showers propagated according to parameters averaged over many events
  - faster than detailed shower simulation

- Transportation
  - only one process which is responsible for moving the particle through the geometry
  - must be assigned to each particle type

Physics Processes

- All the work of particle decays and interactions is done by processes
- A process does two things:
  - decides when and where an interaction will occur
    - method: GetPhysicalInteractionLength()
    - this requires a cross section or decay lifetime
    - for the transportation process the distance to the nearest object along the track is required
  - generates the final state of the interaction (changes momentum, generates secondaries, etc.)
    - method: DoIt()
    - this requires a model of the physics

Physics Processes

- There are three flavors of processes:
  - well-located in space -> PostStep
  - distributed in space -> AlongStep
  - well-located in time -> AtRest
- A process may be a combination of all three of the above
  - in that case six methods must be implemented , one GetPhysicalInteractionLength() and one DoIt() for each type
- "Shortcut" processes are defined which invoke only one
  - Discrete process (has only PostStep physics)
  - Continuous process (has only AlongStep physics)
  - AtRest process (has only AtRest physics)

Example Processes

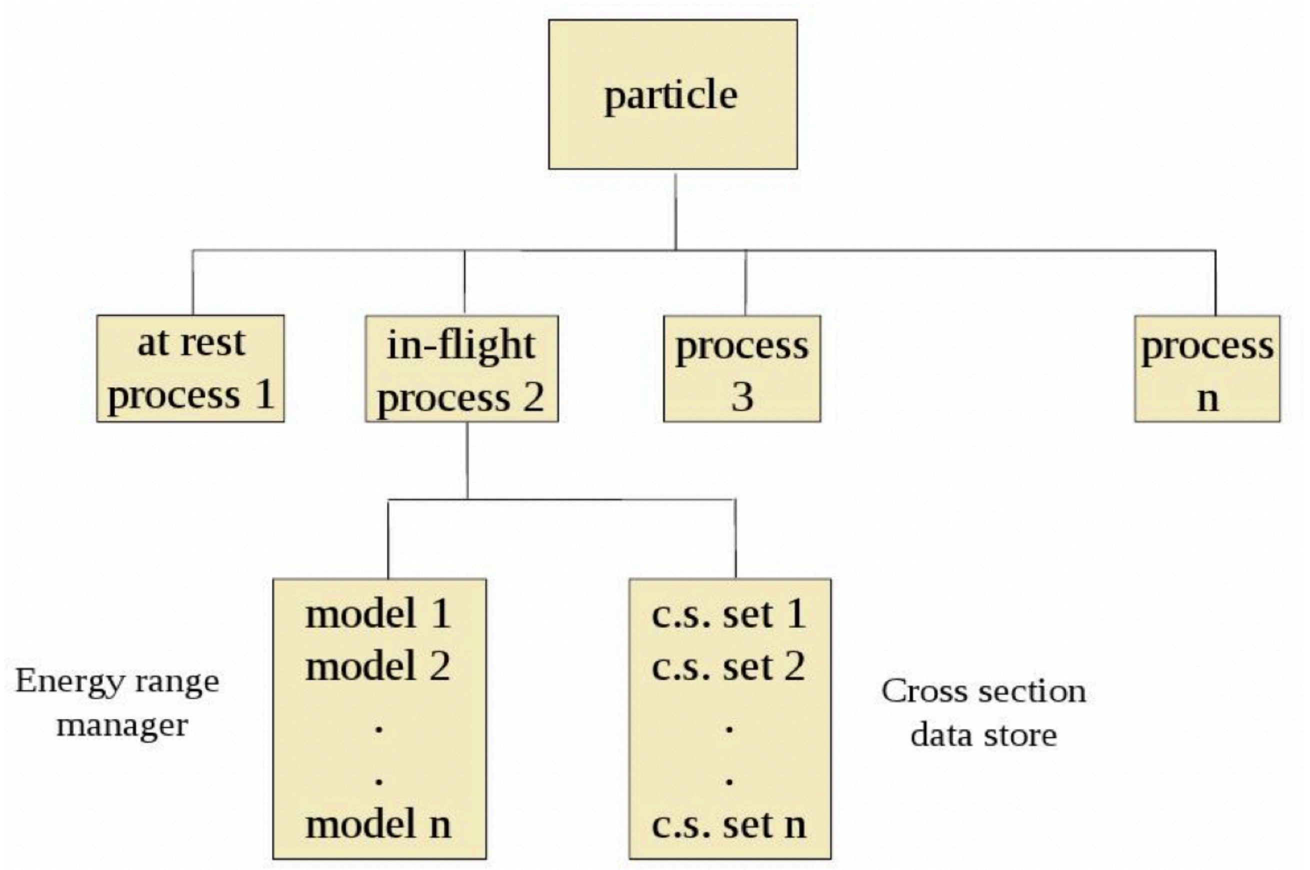
- Discrete process: Compton scattering
  - length of step determined by cross section, interaction at end of step
    - PostStepGetPhysicalInteractionLength()
    - PostStepDoIt()
- Continuous process: Cerenkov effect
  - photons created along step, # proportional to step length
    - AlongStepGetPhysicalInteractionLength()
    - AlongStepDoIt()
- At rest process: μ^- capture at rest
  - muon has already stopped so time is the relevant variable
    - AtRestGetPhysicalInteractionLength()
    - AtRestDoIt()

Example Processes

- Continuous + discrete: ionization
  - energy loss is continuous (low energy electrons not generated)
  - Moller/Bhabha scattering and knock-on electrons are discrete
- Continuous + discrete: bremsstrahlung
  - energy loss is continuous (soft photon emission not performed)
  - hard photon emission is discrete
- Discrete + at-rest: e^+ annihilation
  - in-flight annihilation is discrete
  - at rest annihilation when positron stops
- Multiple scattering is also continuous + discrete

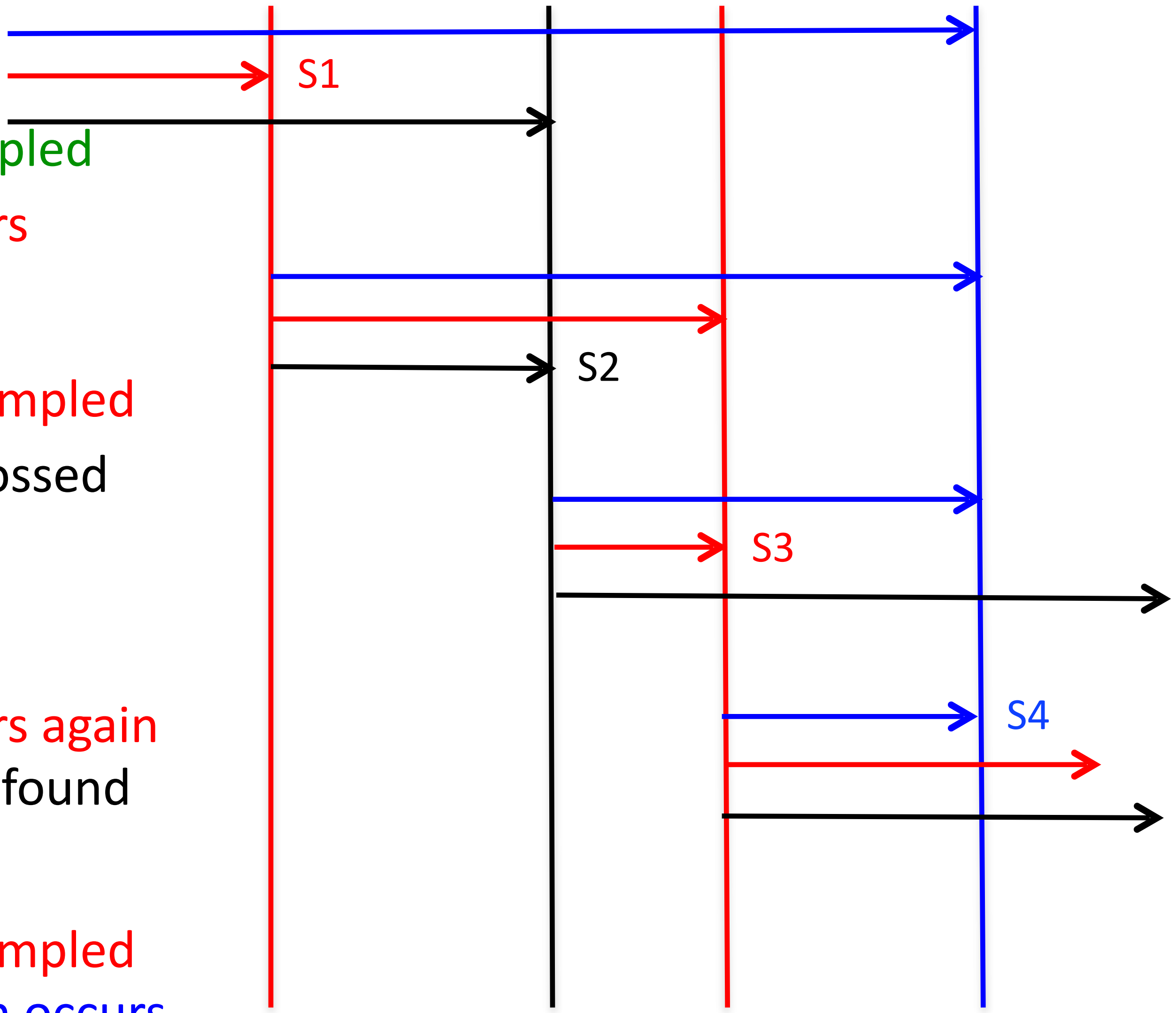
Handling Multiple Processes

- Many processes (and therefore many interactions) may be assigned to the same particle



Handling Multiple Processes

- Step 1:
  - all lengths sampled
  - Compton occurs
- Step 2:
  - Compton re-sampled
  - boundary is crossed
- Step 3:
  - Compton occurs again
  - new boundary found
- Step 4:
  - Compton re-sampled
  - pair production occurs



Summary

- All particles, physics processes and production cuts needed for a specific application must be defined in a physics list

- Two kinds of physics list interfaces are available for users:
  - G4VUserPhysicsList – for relatively simple physics environments
  - G4VModularPhysicsList – for more complex physics environments

- Some reference physics lists are provided by Geant4 developers which may be used as starting points
  - pure EM physics constructors
  - complete hadronic, EM and extra physics

Summary

- Choosing the appropriate physics list for a given application requires care and validation
- Processes handle all the physics of particle interactions
- Geant4 provides processes to cover nearly all particles over energies ranging from 0 to ~TeV
  - users may define their own processes
- Many processes may be assigned to a particle type